Communicating with ALOHA

Draft 4/28/98 -- Jerry Muhasky

This document is for programmers and describes how to program your application to communicate with ALOHA. There are two common types of applications that wish to communicate with ALOHA.

(1) Mapping Applications can draw ALOHA's footprint and communicate conc/dose point information. Section 1 of this document describes how to read ALOHA's reading ALOHA's footprint "status" file, how to communicate a user specified conc/dose point location to ALOHA, and how to ask ALOHA to notify your application about new footprints and conc/dose locations.

(2) Meteorological Applications can communicate meteorological data directly to ALOHA rather than have ALOHA take over the serial port. Section 2 of this document describes the messages and formats for sending atmospheric information to ALOHA.

Microsoft Windows Version: Communication via NOAA_16.DLL

The process of inter-application communication on the IBM is accomplished through the NOAA-written 16 bit DLL called NOAA_16.DLL. (Note: DDE was abandoned in favor of the DLL when DDE proved unreliable). To communicate with other applications, you will need to use the following basic calls. (See the Appendix 3 and 4 for more details.)

(1) **Register** with the DLL by calling NERegister().
(2) **Send messages** by calling NESendMessage() with the message string you wish to send. See Appendix 2 for the details of message strings.
(3) **Receive messages** by calling NEGetNextMessage(). It is recommended that you call this function on idle An alternative method is described in the note following NERegister() in appendix 3.
(4) When quitting your application, it is polite to say 'BYE ' to those applications you have sent messages to, and then unregister with the DLL by calling NEBye().

Macintosh Version: Communication via Apple Events

The process of inter-application communication on the Macintosh is accomplished through Apple Events. All messages are sent with event class 'NOAA' and apple event ID 'AEVT'. We call these special apple events "NOAA messages". Parameters for these NOAA messages are just Apple Event parameters and so are characterized by their 4 char OSType keyword. The data for each keyword parameter is passed as typeChar data. Which keyword parameters are present varies depending on the message, but the '**MSSG**' keyword parameter is required and can be thought of as the message type. We use the phrase an "HOLA message" for a class 'NOAA' apple event which has the value of "HOLA" in parameter specified by the keyword 'MSSG' .

Your program will need to Install an apple event handler to deal with 'NOAA' messages using something like

```
AEInstallEventHandler(OSTYPE_NOAA,'AEVT',
      NewAEEventHandlerProc(HandleNOAAEvent), 0, false);
```

and must retrieve parameters using a function using code like

```
AEGetParamPtr(ae, keyWord, type, actualType, ptr, maxSize, actualSize);
```

where type = typeChar and keyWord is a 4 char code such as 'MSSG'.

Section 1: ALOHA's Footprint and Conc/Dose Point Status

While running, ALOHA maintains a file giving the status of the footprint . ALOHA deletes this file as soon as the information (i.e. footprint) is no longer valid. Your application should not display a footprint whenever the file does not exist. The footprint file is described in the Appendix.

The coordinates of ALOHA's Conc/Dose Point can be requested from ALOHA using inter-application communication as described later in this document. Your application can also ask ALOHA to use your values for the Conc/Dose point coordinates using inter-application communication.

Note: The IBM version of ALOHA maintains a Conc/Dose point information file (is similar to the footprint file, which contains only one line with two numbers in it, representing <Meters East> < Meters North> from the source point. It is called "ALO_CLP.PAS"

Requesting Notification from ALOHA

In order to request notification messages from ALOHA, you must do the following:
(1) Register your application with NOAA_16.DLL (if using MS Windows version)
(2) Send ALOHA a notification request (NFTY message)
(3) remember to say 'BYE ' to ALOHA when quitting.

Sending ALOHA a notification request

To ask ALOHA notify your application when footprint or conc/dose point coordinates change, send a 'REGA' (register application) message to ALOHA and include a 'NTFY' parameter with any value starting with "y" (such as "YES"). As a short cut, you can simply add the 'NTFY' parameter to an 'HOLA' or 'OKHI'. See the appendix 2 for a specific example of the register message.

Terminating and restarting notifications

To unregister, send a 'REGA' messages with 'NO' as the 'NTFY' parameter. To register, just resend a 'REGA' messages with 'YES' as the 'NTFY' parameter.

Messages you will receive from ALOHA

'BYE ' -- when ALOHA quits (note: there is a space character in this 4 char message)

'NTF!' -- when the footprint has changed, ALOHA will send you a 'NTF!' message with the following parameters

'YRPS' -- your PSIG

'EDIS' -- in theory, either "Y" or "N" indicating if the dispersion menu is enabled (i.e., if ALOHA has a source set and the user can choose the footprint menu). But ALOHA 5.2 has a bug where this parameter is always returned as "N". This will be fixed in the next version of ALOHA.

If there is a footprint, the '**FILE**' parameter gives the path name to the ALOHA footprint file

If a valid conc/dose point has been specified, the following parameters will be included

'MTRN' --- the meters north for the conc/dose point

'MTRE' --- the meters east for the conc/dose point

Other optional messages to ALOHA

At anytime, you can send 'NTF?' to request the notification information from ALOHA. Note, you need not have registered with ALOHA in order to do this.

Section 2: Meteorological Information

Meteorological venders who find ALOHA's serial port implementation limiting can write applications that communicate SAM information directly to ALOHA.

In order to use your application as the meteorological information provider for ALOHA, you must do the following:
(1) Register your application with NOAA_16.DLL (if using MS Windows version).
(2) Register your application with ALOHA by sending a 'REGA" message.
(3) Inspect messages sent by ALOHA to know if ALOHA is accepting SAM data.
(4) Send SAM information to ALOHA using '**SAM!** ' messages when ALOHA is accepting SAM data.
(5) Remember to say '**BYE** ' to ALOHA when quitting.

Registering your application with ALOHA

Send ALOHA a '**REGA**' (register application) message and include a '**REGA**' parameter with value of '**SAMA**' (SAM application) indicating your application can communicate SAM information to ALOHA. If ALOHA has received this message before the user selects the SAM menu item, the user is asked if they want to use your application instead of the serial port. If they chose your application, ALOHA will not read from the serial port, but will instead read the information from the **'SAM!'** messages it receives.

Messages you will receive from ALOHA

'SAM!' -- a message to indicate your application should start or stop transmitting data. You will receive this message with "START" in the '**DATA**' parameter after the user selects the SAM menu item to specify that that SAM station should be used. Your application will receive this message with "STOP" in the '**DATA**' parameter when the user switches back to user inputted atmospheric data.

'BYE ' -- when ALOHA quits (note: there is a space character in this 4 char message)

Sending Meteorological Data to ALOHA

The **'SAM!'** messages your application sends to ALOHA simply substitute for reading from the serial port. The data string sent in the '**DATA**' parameter is treated by ALOHA exactly the same as if the string was received through the serial port. In particular, ALOHA expects the same format as used for SAM transmissions to the serial port.

Appendix 1: ALOHA's Footprint file

The footprint (pass) file is called "ALO_FTP.PAS" and contains drawing instructions (in meters east and north) using move to, line to, and arc commands. The footprint and the confidence lines are delineated via keywords FOOTPRINT and CONFIDENCE LINES. The rest of the lines are distinguished by the first letter of the line .

T -- simple text displayed in CAMEO DOS
t -- the text aloha would add to the top of a printout of the footprint
A < p > < q > < r > -- draw an arc from an angle p to the angle q at a radius of r meters.
The angles p and q are expressed somewhat weirdly.
The angles are measured in degrees counterclockwise from the positive x axis.
The arc should be drawn in a counterclockwise direction from p to q.
ALOHA does not make any effort to use the interval from 0 to 360 degrees.
M < x > < y > -- move to x y (in meters east, meters north from the source) (pen up)
L < x > < y > -- draw line to x y (in meters east, meters north from the source)

An example ALOHA footprint file would contain text like:

```
T ALOHA heavy gas footprint
T CHLORINE
T LOC: IDLH (10 ppm)
FOOTPRINT
A -5.0 175.0 13.6
M -13.6 1.2
L -14.4 -9.0
<more line to lines>
L 13.6 -1.2
CONFIDENCE LINES
M -13.0 8.0
L -17.2 -1.9
<more line to lines>
L -1588.1 -3009.2
A -117.8 -95.0 3401.5
A -95.0 -72.2 3401.5
M 1041.4 -3239.2
L 1145.0 -2913.7
<more line to lines>
L 14.1 5.6
t  Time: July 30, 1996 & 1828 hours EDT (Using computer's clock)
t  Chemical Name: CHLORINE
t  Wind: 5 knots from 5° true at 3 meters
t  FOOTPRINT INFORMATION:
t    Model Run: Heavy Gas
t    User-specified LOC: equals IDLH (10 ppm)
t    Max Threat Zone for LOC: 2.1 miles
```

Converting file values to Latitude and Longitude

The following code fragments from ALOHA may prove useful.

```
/*-------------------------------------------*/
void ExampleDrawPlume(void)
{
   short i;
   FPairPtr LL = (FPairPtr)NewPtrClear(1000 * sizeof(FPair));

   if(LL== nil)
   {
      if(LL)DisposPtr((Ptr)LL); LL = nil;
      SysBeep(5);
      // memory error
      return ;
   }

   if (!ConvertPlume(LL))
   {
      // there is no plume
      // make sure MARPLOT is not showing one
      DeleteAlohaObjects(IAC_CONFIDENCE_LINES + IAC_PLUME_POLYGON);
      if(LL)DisposPtr((Ptr)LL); LL = nil;
      return ;
   }

   ////////////////////////////
   // after calling ConvertPlume() the structure LL
   // is filled in with the lat,long values
   //  (i = 0 ; i < firstConfidencePoint ; i++)
   // defines the FOOTPRINT
   //  (i = firstConfidencePoint ; i < sNumPoints ; i++)
   // defines the CONFIDENCE POLYGON
   ////////////////////////////

   /////////////////
   // YOUR CODE GOES HERE
   /////////////////
   // draw around the footprint, then the confidence lines
   /////////////////

   if(LL)DisposPtr((Ptr)LL); LL = nil;
   return true;
/*-------------------------------------------*/

typedef struct
{
   float x;
   float y;
} FPair;
typedef FPair FAR *  FPairPtr;

static short sNumPoints, firstConfidencePoint;
```

```
/*-----------------------------------------------*/
double LongToLatRatio(double baseLat)
{ return cos(baseLat * PI/180); }

double MilesPerDegreeLong(double baseLat)
{ return 69 * LongToLatRatio(baseLat); }

double MetersPerDegreeLong(double baseLat)
{ return MILESTOMETERS * MilesPerDegreeLong(baseLat); }

double DegreesLongPerMeter(double baseLat)
{ return 1 / MetersPerDegreeLong(baseLat); }
/*-----------------------------------------------*/
void AddArcPoints(float arcStartAngle,float arcEndAngle,float arcRadius,FPairPtr XY)
{
   OSErr err = noErr;
   float x, y, angle;

   if(XY == nil) { ProgrammerError(); return; }
   arcStartAngle = fmod(arcStartAngle, 360);
   arcEndAngle = fmod(arcEndAngle, 360);
   if (arcStartAngle < 0) arcStartAngle += 360;
   if (arcEndAngle < 0) arcEndAngle += 360;
   if (arcStartAngle > arcEndAngle) arcEndAngle += 360;
   // arbitrarily choose to mark every 10 degrees
   for (angle = arcStartAngle ; angle < arcEndAngle ; angle += 10) {
      x = cos(angle * (PI/180)) * arcRadius;
      y = sin(angle * (PI/180)) * arcRadius;
      XY[sNumPoints].x = x;
      XY[sNumPoints].y = y;
      sNumPoints++;
   }
   x = cos(arcEndAngle * (PI/180)) * arcRadius;
   y = sin(arcEndAngle * (PI/180)) * arcRadius;
   XY[sNumPoints].x = x;
   XY[sNumPoints].y = y;
   sNumPoints++;
}
/*-----------------------------------------------*/
void ClosePolyAlways(FPairPtr XY)
{
   // close the polygon in all cases
   // note: firstConfidencePoint is zero when we close the footprint
   // and is positive when we close the confidence lines, so we can use it as
   // the beginning of our poly in both cases
   double streetMetricDist = fabs(XY[firstConfidencePoint].x -XY[sNumPoints-1].x)

   +fabs(XY[firstConfidencePoint].y -XY[sNumPoints-1].y);
   if( 0 < streetMetricDist && firstConfidencePoint < sNumPoints)
   {
      // point is different
      // add the point
      XY[sNumPoints].x = XY[firstConfidencePoint].x;
      XY[sNumPoints].y = XY[firstConfidencePoint].y;
      sNumPoints++;
   }
}
```

```
/*-------------------------------------------*/
Boolean ConvertPlume(FPairPtr XY)
{
   char miniBuffer[100];
   float x, y, arcStartAngle, arcEndAngle, arcRadius;
   short i, fRef;
   OSErr err = noErr;
   float theLat,theLng;
   char fileName[256];

   long fileSize = 9999, position = 0;
   Boolean done = false;
   Ptr buffer = NewPtrClear(10000);

  #ifdef IBM
      OFSTRUCT localOFStruct;
      memset(&localOFStruct,0,sizeof(OFSTRUCT));
      localOFStruct.cBytes = sizeof(OFSTRUCT);
  #endif

   if( XY == nil || buffer == nil)
   {
      // not enough memory
      if(buffer)DisposPtr(buffer); buffer = nil;
      SysBeep(5);
      return false;
   }

   OurDirectoryWithDelimiter(fileName);
   strcat(fileName,"ALO_FTP.PAS");

   // read in pass file, writing meter offsets into array
   //and converting arcs to meter arrays
  #ifdef IBM
    fRef = (short)OpenFile (fileName, &localOFStruct, OF_READ|OF_SHARE_DENY_WRITE);
   if (fRef == -1) err = true;;
  #else
   err = fsopen(fileName, 0, &fRef);
  #endif

   if(err)
   {
      if(buffer)DisposPtr(buffer); buffer = nil;
      return false;
   }
   FSRead(fRef, &fileSize, buffer);
   FSClose(fRef);
   buffer[9999] = RETURN_CHAR;

   sNumPoints = 0;
   firstConfidencePoint = 0;
```

```
while (position < fileSize) {
      switch (buffer[position]) {
         case 'T': case 'F': break;
         case 'C':
            ClosePolyAlways(XY);// close footprint
            firstConfidencePoint = sNumPoints;
            break;
         case 'L': case 'M':
            while (buffer[position] != ' ') position++;
            strncpy(miniBuffer, &buffer[position], 99);
            miniBuffer[99] = 0;
            sscanf(miniBuffer, "%f %f", &x, &y);
            XY[sNumPoints].x = x;
            XY[sNumPoints].y = y;
            sNumPoints++;
            break;
         case 'A':
            while (buffer[position] != ' ') position++;
            strncpy(miniBuffer, &buffer[position], 99);
            miniBuffer[99] = 0;
            sscanf(miniBuffer,"%f %f %f",&arcStartAngle,&arcEndAngle,&arcRadius);
            AddArcPoints(arcStartAngle, arcEndAngle, arcRadius,XY);
            break;
      }
      while (buffer[position] != RETURN_CHAR) position++;
      position++;
      if(buffer[position] == LINEFEED_CHAR) position++;
      // go past the LINEFEED_CHAR as well
   }

   ClosePolyAlways(XY);// close confidence lines
   for (i = 0 ; i < sNumPoints ; i++) {
      theLat =  sourceLat + XY[i].y * DEGREESLATPERMETER;
      theLng =  sourceLng - XY[i].x * DegreesLongPerMeter(sourceLat);
      // note: the order, x is lat, y is long
      XY[i].x = theLat;
      XY[i].y = theLng;
   }

   ////////////////////////////////////////////
   if(buffer)DisposPtr(buffer); buffer = nil;
   return true;
}
```

Appendix 2: Message String Format

Messages are based on 4 character "messages" and 4 character "parameter keys".
The parameters or keys are separated by a vertical tab (ascii char 11) which is denoted as <vt> below. Message keys are all capitals (and case sensitive).

Your application needs to identify itself with a 4 character "signature" and a 4 character "psuedoSignature".

Note: On the Macintosh, applications have a hidden file attribute , called the signature that the system uses when to match files with their creator. The psuedoSignature allows a single executable to have several different "identities" on the Macintosh where the signature is fixed and predetermined for such applications as HyperCard and FoxPro.

On the IBM, you are free to choose any signature and psuedoSignature you like.
ALOHA's signature is ALH5.

On the IBM, message are one long c-string of keys and values.

The first few keys are required and must be in this order.
MSSG = the message key (in the example below, let's use "REGA" register application)
SIGN = your signature (in the example below, let's use "FRED")
PSIG = your psuedoSignature (in the example below, lets use "PETE")
XTRA = an extra string (you can usually pass "")

These required values are followed by the optional parameter keys and values.
Different messages require different parameters. The order of the optional parameters is not specified and it is recommended you put the short parameters toward the front of the message string to speed parsing.

Example messages to ALOHA (which has a signature of "ALH5".)
Call NESendMessage("ALH5",messageStringBelow,FALSE,NULL,NULL));
with the messageStringBelow.

(1) To register you application with ALOHA, send this message to 'ALH5'

```
MSSG<vt>REGA<vt>SIGN<vt>FRED<vt>PSIG<vt>PETE<vt>XTRA<vt><vt>NTFY<vt>YES
```

(2) To get ALOHA to use your values for the Conc/Dose point, send ALOHA ('ALH5') a 'CDP!' message with the meters east 'MTRE' value and meters north 'MTRN' value.

```
MSSG<vt>CDP!<vt>SIGN<vt>FRED<vt>PSIG<vt>PETE<vt>XTRA<vt><vt>MTRE<vt>35.2
<vt>MTRN<vt>56.89
```

An example of a notification message ('NTF!') you will receive from ALOHA is

```
MSSG<vt>NTF!<vt>SIGN<vt>ALH5<vt>PSIG<vt>ALHA<vt>XTRA<vt><vt>YRPS<vt>PETE
<vt>EDIS<vt>Y<vt>MTRE<vt>1000<vt>MTRN<vt>-23.6<vt>FILE<vt>C:\ALOHA\ALO_FTP.PAS
```

Appendix 3: NOAA_16.DLL

```
long FAR PASCAL _export  NERegister(
Ptr sig,  // 4 char string to identify your application, e.g.ALOHA is "ALHA"
HWND mainHWnd,
Ptr className, // the class name of your application (if known)
Ptr messageStringForHola, // unimplemented, pass NULL
Ptr humanName, //to specify your application, e.g.ALOHA uses "ALOHA"
Ptr wakeUpTopicString,
Ptr fullPath, // of your application
UINT wakeUpMessage,// the message  parameter to be used in SendMessage()
WORD wakeUpWord, // the WORD parameter to be used in SendMessage()
LONG wakeUpLong // the LONG parameter to be used in SendMessage()
);
```

Important Note: If you specify an non-zero wakeUpMessage parameter, then whenthe DLL recieves a message for your application, it will send your main window handle a message using the Window's SendMessage() function. Pass zero for wakeUpMessage if you do not want a message sent to your main window handle. We recommend avoiding the use of a wakeup message when you are able to check for your messages on idle. Checking for messages on idle is a very safe way to handle the whole situation. If you do request a wakeup message, be certain to code for the possibility that the DLL will send you that wakeup message again (for another message) while your code is handling the first message. We had trouble with this in Foxpro. Finally, as a special case to support CAMEO Windows (which was written in Foxpro 2.6), if the wakeUpMessage is WM_COMMAND, then the parameters have special meaning and the DLL will send a message corresponding to selecting a menu item. For details, see Appendix 4.

```
long FAR PASCAL _export  NEBye(Ptr sig,
HWND mainWHnd,
Ptr className,
Ptr messageStringForBye  // unimplemented, pass NULL);

long  FAR PASCAL _export NESendMessage(
Ptr toSig, // the 4 charater code of the application you want to talk to
Ptr messageString,
UINT launch, // unimplemented, pass FALSE
Ptr humanName,  // unimplemented, pass NULL
Ptr possibleFullPath  // unimplemented, pass NULL);

Boolean FAR PASCAL _export  NEAppIsRunning(Ptr sig);

long  FAR PASCAL _export   NEGetNextMessageLength(Ptr sig);

Boolean  FAR PASCAL _export  NEGetNextMessage(
Ptr sig, Ptr s,
long maxLength);
```

// returns TRUE if it successfully filled in your parameter

```
long  FAR PASCAL _export  NEPeekMessage(
int n, Ptr sig,Ptr s,
long maxLength,long offsetIntoMessage);
```

// returns the length of the next message, so you can allocate enough space. Your application can probably just allocate 256 bytes and forget about it.

Appendix 4: Our FoxPro Wakeup Implementation

Motivation:

Because FoxPro 2.6 did not support an "on idle" hook, we needed a way to get FoxPro's attention when the NOAA_16.DLL had a message for FoxPro. Our solution was to send the main window handle a message simulating the selection of a menu item. The selection of a menu item is a supported way to get FoxPro to execute our script to retrieve and handle the message. he only problem we had to overcome was the fact that we did not know the ID of the menu item because the FoxPro application creates the menu and assigns the ID.

To give the exact example used in CAMEO, we installed our wakeup menu under a "Sharing" Menu which had a hiarchical submenu for "MARPLOT" which had the wakeup menu item as an item. The menus look something like:

```
File   Edit   Sharing
         ALOHA
         MARPLOT
            Get Info
            Link
            -(the wakeup menu that executes the code in CAMEO)
            Go to MARPLOT
         SitePlan
```

We had control over all of the items under the Sharing menu , so we could rely on the position of the wakeup menu item under the Sharing menu. We solved the problem of finding the main menu "group" by passing the text of the main menu (e.g. "Sharing") as a parameter. To find the submenu item, we pass numbers for the position.

Implementation:

To request that NOAA_16.DLL send your application simulating a menu item being selected when you have a message, you need to call NERegister() with the following parameters.

wakeUpTopicString -- the text of the menu your menuitem is under. In the case of CAMEO this was "Sharing".
wakeupMessage -- WM_COMMAND (= 256+16+1)
wakeupWord -- itemNum of the first submenu (In the example shown above, this would be the 0-relative number of the MARPLOT menu, which is 1 (**)
wakeupLong -- subitemNum of the wakeup menu item. In the example shown above, this would be 2 (= 3-1) (**).

(**) Note: the itemNum and subitemNum are 0-relative positions. I.e., the counting starts with 0, so the first position is number 0, the second position is number 1, etc.

Note: if your wakeup menu item is not under a hiarchtical submenu but is directly under the main menu "group", pass 0 for the subitemNum.

For specifics on what the DLL code actually does, you can examine the function below which is used by the DLL to find the menu number.

(code from NOAA_16.DLL)

```
WORD GetMagicMenuID(HWND hWnd,Ptr topicName, int itemNum, int subitemNum)
{
   int i,len;
   HMENU mainMenu = GetMenu(hWnd);
   HMENU topicMenu;
   HMENU itemMenu;
   char str[64];
   WORD theID = 0;

   //for( topicMenu = 1,i = 4 ;i < 15 && mainMenu && topicMenu; i++)
   for( topicMenu = 1,i = 1 ;i < 15 && mainMenu && topicMenu; i++)
   {
      // look for the sharing menu
      len = GetMenuString(mainMenu,i,str,64,MF_BYPOSITION);
      if(StrMatches(str,topicName))
      {
         topicMenu = GetSubMenu(mainMenu,i);
         itemMenu = GetSubMenu(topicMenu,itemNum);
         if(subitemNum == 0)
         {
            theID =  GetMenuItemID(topicMenu,itemNum);
         }
         else
         {
            ////////////////////
            theID = GetMenuItemID(itemMenu,subitemNum);
         }
         break;

      }
   }
   return theID;
}
```

Appendix 5: IBM Code from ALOHA

```
#ifdef IBM ///{
HINSTANCE gNoaaDllInst;
Ptr gDllFileName ="NOAA_16.DLL";

void CallNERegister(void)
{
   char sigStr[6];
   char fullPath[256];
   char humanName[64];
   FARPROC proc=NULL;
   if(gNoaaDllInst == NULL)
   {
      gNoaaDllInst = LoadLibrary(gDllFileName);
   }
   if((UINT)gNoaaDllInst > 32)
   {
      //we have the library
      proc = GetProcAddress(gNoaaDllInst,"NERegister");
      if(proc)
      {
         OSType2String(gMySignature,sigStr);
         GetPathToAlohaPlusExtension(fullPath);
         getindstring(humanName,1000,1); //ALOHA
         (*proc)( (Ptr) sigStr,
                  (HWND) global_stdglobal_ptr->main_wnd_hdl,
                  (Ptr) MAIN_CLASS_NAME,
                  (Ptr) NULL, //messageStringForHola,
                  (Ptr) humanName,
                  (Ptr) "ALOHA.exe",
                  (Ptr) fullPath,
                  (UINT) SA_APPTASK,
                  (WORD) 0,
                  (LONG)0);
      }
   }
}
void CallNEBye(void)
{
   char sigStr[6];
   FARPROC proc=NULL;
   if((UINT)gNoaaDllInst > 32)
   {     //we have the library
      proc = GetProcAddress(gNoaaDllInst,"NEBye");
      if(proc)
      {
         OSType2String(gMySignature,sigStr);
         (*proc)( (Ptr) sigStr,
                  (HWND) global_stdglobal_ptr->main_wnd_hdl,
                  (Ptr) MAIN_CLASS_NAME,
                  (Ptr) NULL); //messageStringForBye,
      }
      FreeLibrary(gNoaaDllInst);
      gNoaaDllInst = NULL;
   }
}
```

```
OSErr CallNESendMessage(Ptr toSigStr, Ptr messageStr)
{
   FARPROC proc=NULL;
   OSErr err = -1;
   if((UINT)gNoaaDllInst > 32)
   {  //we have the library
      proc = GetProcAddress(gNoaaDllInst,"NESendMessage");
      if(proc) err = (OSErr)(*proc)((Ptr)toSigStr,(Ptr)messageStr,
            (Boolean)FALSE,(Ptr)NULL,(Ptr)NULL);
   }
   return err;
 }
Boolean CallNEAppIsRunning(Ptr toSigStr)
{
   FARPROC proc=NULL;
   Boolean isRunning = FALSE;
   if(gNoaaDllInst == NULL)  gNoaaDllInst = LoadLibrary(gDllFileName);
   if((UINT)gNoaaDllInst > 32)
   {  //we have the library
      proc = GetProcAddress(gNoaaDllInst,"NEAppIsRunning");
      if(proc) isRunning = (Boolean)(*proc)((Ptr)toSigStr);
   }
   return isRunning;
 }
OSErr HandleNEMessage(void)
{  // check for a message and handle it
      char sigStr[6];
   FARPROC proc=NULL;
   OSErr err = -1;
   long len;
   if(gNoaaDllInst == NULL)  gNoaaDllInst = LoadLibrary(gDllFileName);
   if((UINT)gNoaaDllInst > 32)
   {  //we have the library
      OSType2String(gMySignature,sigStr);
      proc = GetProcAddress(gNoaaDllInst,"NEGetNextMessageLength");
      if(proc){
         len = (long)(*proc)((Ptr)sigStr);
         if(len > 0)
         {  // we have a message
            proc = GetProcAddress(gNoaaDllInst,"NEGetNextMessage");
            if(proc){
               long maxLength = len+1;
               Ptr messageString = NewPtrClear(maxLength);
               if(messageString== nil) MessageBeep(5);// memory error
               else{
                  Boolean gotIt;
                  gotIt = (Boolean)(*proc)((Ptr)sigStr,
                  (Ptr)messageString,(long)maxLength);
                  if(gotIt) HandleNOAAEvent(&messageString,nil,0);
                  else MessageBeep(5);
                  DisposPtr(messageString);
               }
            }
         }
      }
   }
   return err;
 }
```

Appendix 6: Other Messages To/From ALOHA

Messages to ALOHA

(all messages include 'SIGN', 'PSIG', 'MSSG' and 'XTRA' parameters)

Message	Parameters	Description
'BYE '		The friend application is quitting. ALOHA will not send any more messsages to it.
'HOLA'		Initial greeting from a friend application. The friend sends this message to ALOHA when it starts up and sees that ALOHA is running. This tells ALOHA that the friend is alive and ready to handle messages. ALOHA responds with an OKHI message. If the sender is MARPLOT, ALOHA also sends a MENU message to install ALOHA's sharing menu.
	'VERS'	"2" or greater means you are using the updated IAC messages
	'NAME'	name of the friend application
	'PATH'	full path to friend applications's executable file
	'DOC '	(optional) full path to default document to be opened when ALOHA launches the friend
'OKHI'		Acknowledge receipt of HOLA from ALOHA. The friend application has received an HOLA message from ALOHA, and is acknowledging so that ALOHAwill know the friend is running. If the sender is MARPLOT, ALOHA also sends a MENU message to install ALOHA's sharing menu.
	'VERS'	"2" or greater means you are using the updated IAC messages
	'NAME'	name of the friend application
	'PATH'	full path to the friend application's executable file
	'DOC '	(optional) full path to default document to be opened when ALOHA launches the friend

'MENU'		Install sub-menu in ALOHA's Sharing menu. This adds a new sub-menu to ALOHA's Sharing menu. If a sub-menu with the same name already exists, it is replaced with the new menu. (Thus, a friend application can send a MENU message each time it greets ALOHA, without worrying about duplicating the menu.) Menus installed in ALOHA are automatically saved by ALOHA. They can be used later, even when the friend application is not running. In this case, ALOHA launches the friend application before sending it the MHIT message.
	'VERS'	"2" or greater; needed to show ALOHA you are using the updated IAC messages
	'NAME'	name of menu
	'ITMS'	return-delimited string of menu items
	'PATH'	full path to the friend application's executable file
	'DOC '	(optional) full path to default document to be opened when the friend is launched
'FRWD'		Your application can ask ALOHA to bring it to the foreground using this message. There are a number of technical issues involved in getting an application to come automatically to the foreground. These issues are different for each platform/system. In some cases, when an application wants to bring some application (often itself) to the foreground, it is easier (and sometimes more polite) to ask another application to do the job.
	'WHO '	signature of application to bring forward (usually the sending app itself)
	'NAME'	name of application to bring forward; on Windows, NAME should be the title of your main window, or the name of your main window class.
		NOTE: On the Macintosh, it is sometimes better to use the Notification Manager and let the user bring you forward.
'CHM?'		A request for ALOHA to send a CHM! message indicating the current ALOHA chemical.
'CHEM'		A request for ALOHA to select a chemical.
	'NOAA'	(optional) A string containing the NOAA number of the chemical.
	'CAS '	(optional) A string containing the CAS number of the chemical (as a number with no dashes).
	'NAME'	(optional) A string containing the name of the chemical.

Other messages sent to ALOHA by MARPLOT:

MHIT	ALOHA assumes it is it's menu in MARPLOT but does not check the sender
CPNT	Marplot click point response. ALOHA assumes that the sender was MARPLOT but that is not checked
OVL!'	ALOHA simply records the ID
OBID	ALOHA just records the IDS
INFO	ALOHA checks that the sender is marplot then checks to see if there is one object and it's data matches CONCDOSE_POINT if so shows the conc window , else it shows the text summary
CLOS	If from MARPLOT, ALOHA clears the globalID's
ULK?	ALOHA sends MARPLOT an alert saying you can't unlink ALOHA objects

Messages From ALOHA

(all messages include 'SIGN' ('ALH5'), 'PSIG' ('ALHA'), 'MSSG' and 'XTRA' parameters)

Message	Parameters	Description
'BYE '		ALOHA is quitting. This is to inform you that if you plan to send ALOHA any more messages, you will have to wait until it gets started again (perhaps by your launching it) and sends you an HOLA message. This message is sent to all friend applications that are currently running.
'HOLA'		Initial greeting from ALOHA. ALOHA sends this message to all running friend applications when it starts up. This tells friend applications that ALOHA is alive and ready to handle messages. ALOHA's friend applications are MARPLOT, CAMEO, and any application that has ever said HOLA to ALOHA . The list of friends is saved in the ALOHA.PRF file.
	'NAME'	"ALOHA"
	'PATH'	full path to ALOHA
	'DOC '	empty string; provided for consistency with other applications
	'VERS'	"2"
	=====>	When you get an HOLA message from ALOHA, you should respond with an OKHI message.
'OKHI'		Acknowledge receipt of HOLA. ALOHA has received an HOLA message from an application that just started up and is acknowledging so the other application will know ALOHA is alive. You should treat an incoming OKHI message the same as an incoming HOLA message; they give the same information but you will get one or the other depending on whether your application or ALOHA is started first.
	'NAME'	"ALOHA"
	'PATH'	full path to ALOHA
	'VERS'	"2"
	'DOC '	empty string; included for consistency with other applications

'FRWD'		A request to bring an application forward. ALOHA will only ask you to bring either your own application forward or to bring ALOHA forward. There are a number of technical issues involved in getting an application to come automatically to the foreground. These issues are different for each platform/system. In some cases, when an application wants to bring some application (often itself) to the foreground, it is easier (and sometimes more polite) to ask another application to do the job.
	'WHO '	signature of application to bring forward
	'NAME'	name of application to bring forward; on Windows, NAME should be the title of your main window, or the name of your main window class.
		NOTE: On the Macintosh, it is sometimes better to use the Notification Manager and let the user bring you forward.
'CHM!'		The answer to a request for ALOHA to indicate which chemical is selected in ALOHA.
	'NOAA'	A string containing the NOAA number of the chemical. (User-added chemicals have 0 for a NOAA number)
	'CAS '	A string containing the CAS number of the chemical (as a number with no dashes). (User-added chemicals have 0 for a CAS number)
	'NAME'	A string containing the name of the chemical.
'RIDS'		Sent to Cameo (CAMO) when the user asks to see RIDS information on the chemical currently selected in ALOHA.
	'NOAA'	A string containing the NOAA number of the chemical. (User-added chemicals have 0 for a NOAA number)
	'CAS '	A string containing the CAS number of the chemical (as a number with no dashes). (User-added chemicals have 0 for a CAS number)
	'NAME'	A string containing the name of the chemical.

Other messages sent to MARPLOT by ALOHA:

CPNT	to get the click point
DELO	to delete our objects
DLOV	to delete our overlay
FRWD	to bring aloha forward and also to bring marplot forward
IMPT	to display the plume etc
MENU	to install a sharing menu in Marplot
MKOV	to create our temp overlay and get the layer ID